SYSC 4810 A — Assignment Report

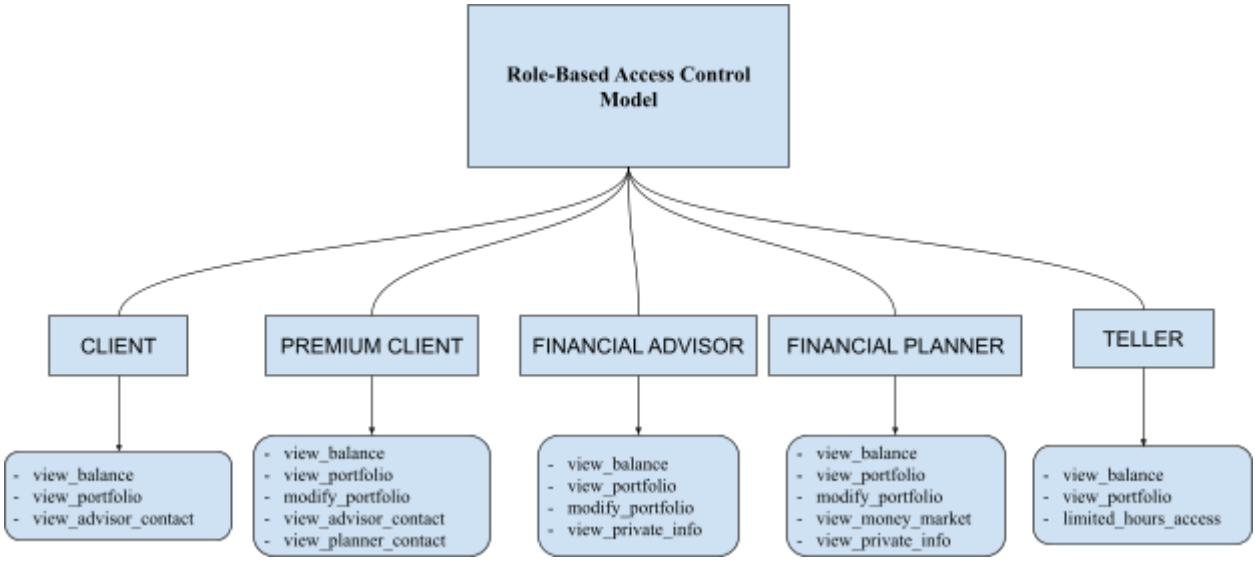
Student: Arun Karki
Student ID: 101219923
Course Instructor: Prof. Hala Assal
Date: December 3, 2024

Problem 1: Access Control Mechanism

(a) Access Control Model Selection

I decided to use the Role-Based Access Control (RBAC) model due to the system's requirements naturally aligning with the principles of the RBAC model. Every role (Client, Premium Client, Financial Advisors, etc.) could be assigned a level of clearance and be mapped to a set of permissions. Additionally, the management of new roles and permissions is easy due to the simplicity of the design pattern.

(b) Access Control Mechanism Design



The figure demonstrates a sketch of the RBAC model in the context of the implementation. Each role is mapped to a set of operations permitted by the system. Visually, we can also see the overlap of certain operations which make sense as premium clients should inherit the operations permitted by a regular client. As demonstrated by the sketch, it should be infeasible for a role to execute operations that are not included in its set of operations. To clarify, a client should not be able to execute the operations permitted by the "FINANCIAL ADVISOR" role as there is no mapping from "CLIENT" to "FINANCIAL ADVISOR."

(c) Implementation Testing

The access control mechanism was implemented to enforce role-specific access policies, ensuring that users can only perform authorized operations.

Test Cases:

1. Test Client Permissions:
    - Verify that clients can only access functions they are authorized for.
    - Tested Functions:
        - view_balance: Expected Result: True
        - view_portfolio: Expected Result: True
        - view_advisor_contact: Expected Result: True
        - modify_portfolio: Expected Result: PermissionError (Access Denied)
2. Test Financial Advisor Permissions:
    - Tested Functions:
        - view_balance: Expected Result: True
        - modify_portfolio: Expected Result: True
        - view_private_info: Expected Result: True
        - view_money_market: Expected Result: PermissionError (Access Denied)
3. Test Financial Planner Permissions:
    - Tested Functions:
        - view_money_market: Expected Result: True
        - view_private_info: Expected Result: True
        - modify_portfolio: Expected Result: True

All tests passed, demonstrating adequate coverage of the access control policy.

Problem 2: Password file

(a) Select the hash function

For the password security implementation, I opted for the SHA-256 hashing algorithm. It is a readily available hashing algorithm from the Python Hashlib library. This algorithm uses 256 bit hash values which makes it incredibly resilient against brute-force attacks. Additionally, the password is appended to a 16-byte salt to ensure a unique hashed value is produced for identical passwords.

(b) Design the password file structure

All passwords are stored in the passwd.txt file and contain the following fields:

username:hashed_password:salt:role

An example of this is:

arun:8c81c070e6e03c157035afc351f9befed3fad98fba19fe08e905f8f8f4bc6b50:835604404529bf49de2acc4bedf6b0d0:client

The first field represents the username, the second field is the hashed password with the salt (TestPass1!), and the third field is the user's role.

(c) Please refer to the main.py file for the full implementation.

(d) Test the password file usage

Test Cases:

1. Salt Generation:
    - Verify the length and uniqueness of the salt.
    - Expected Result: Unique 32-character salts.
2. Password Hashing:
    - Verify that the same password and salt produce consistent hashes.
    - Expected Result: Identical hashes for identical inputs.
3. User Retrieval:
    - Retrieve user details from passwd.txt.
    - Expected Result: Correct username, hashed password, and role.

All tests passed successfully.

Problem 3: Enrol Users

(a) Please refer to the main.py file for the full implementation.

(b) Design and implement the proactive password checker

The password checker enforces the following policy:

- Length between 8 and 12 characters.
- Contains at least one uppercase letter, one lowercase letter, one digit, and one special character (!, @, #, $, %, *, &).
- Does not match the username or appear on a list of common passwords.

(c) Test the enrolment mechanism

Test Cases:

1. Password Length:
    - Input: "short" → Expected: False (too short)
    - Input: "toolongpassword123" → Expected: False (too long)
2. Character Requirements:
    - Input: "TestPass1!" → Expected: True (all requirements met)
3. Common Passwords:
    - Input: "Passw0rd!" (from common_passwords.txt) → Expected: False

All test cases passed and adhered to the policies.

Problem 4: Login Users

(a) Please refer to the main.py file for the full implementation.

(b) Please refer to the main.py file for the full implementation.

(c) Test the user login and access control mechanism

Test cases ensure the login interface works and the user has access to control mechanisms.

Test Cases:

1. Authentication:
    - Correct username and password → Expected: Authentication Successful.
    - Incorrect password → Expected: Authentication Failed.
2. Access Privileges:
    - Display correct access privileges for each role.
    - Example: A Client should see options to view_balance, view_portfolio, and view_advisor_contact.

All tests passed, ensuring proper functionality of the login and access control mechanisms.